CONTROL BREAKS - TAKING INTERMEDIATE TOTALS

24708-2/8507

Printed in South Africa

PRACTICAL PROGRAMMING IN BASIC - SERIAL 24708 ED 2

AMENDMENTS

Please amend the Flowchart solution on page 22 of unit 4708 as follows:

1. In the second column, after the 1st connector symbol, the box

Add 1 to Total Employees

Should be removed and transfered to the fourth column to appear between any of the three boxes shown.

2. In the 4th column, in addition to the foregoing, a new box

Add Sub-total hours to Total hours

must be inserted just before the 'Return to Caller' circle.

SCHOOL OF COMPUTER TRAINING

PROGRAMMING IN BASIC
STUDY UNIT 8

CONTROL BREAKS — TAKING INTERMEDIATE TOTALS

Edition 2

STUDY UNIT 8

YOUR LEARNING OBJECTIVES

WHEN YOU COMPLETE THIS UNIT, YOU WILL BE ABLE TO:

☐ Use intermediate totals logic to generate a sales commission report . . . . . **Pages 1–8**

☐ Apply subroutines to structure program logic . . . . . **Pages 8–13**

☐ Read display output by using the PAUSE statement . . . **Page 13**

☐ Understand a sample program which applies the major logic used in control break processing . . . . . **Pages 13–20**

☐ Provide summarized output when the detail data is not important to the user . . . . . **Pages 20–23**

☐ Sub-divide and total groups of records by using the logic of multiple control breaks . . . . . **Pages 24–28**

☐ Adapt a printed program to account for additional factors . . . . . **Pages 28–30**

STUDY UNIT 8

CONTROL BREAKS—TAKING INTERMEDIATE TOTALS

DO YOU KNOW?

- How records can be processed as control groups?
- The difference between a GOTO and a GOSUB?
- What group printing means?

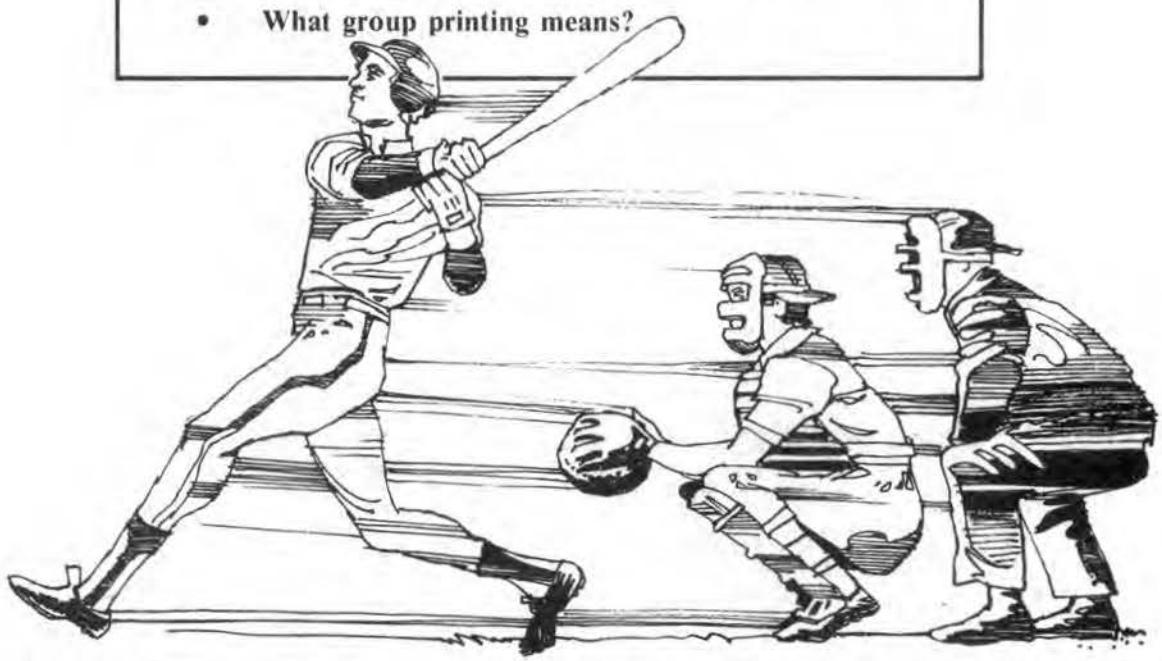


FIGURE 1—Baseball scores and statistics involve taking intermediate totals by half inning, inning, game, and series. No wonder baseball professionals and fans alike welcomed the computer as an aid in keeping all the data instantly available.

THE BREAKS OF THE GAME

Ever think about all the things you learn just from playing different games? Card games, for example, teach us to distinguish values of the various playing cards. A king is higher than a jack. Jacks are higher than any number card in many games. We can even change card values by designating "wild cards" or assigning "penalties" such as the Queen of Spades counting 13 points against you. Once these values are known, scores can be figured.

Keeping score of each hand played, adding this score to the overall total, then coming up with scores for each game played, can get pretty complicated. Bridge players keep score of hands, games, rubbers and sometimes the totals for individual players.

In bowling, the score for each ball is recorded according to the frame and player. Some scores such as a strike or spare count for more than one frame. And in the tenth frame, you get extra chances to score higher by making strikes. Keeping score of two bowling teams of four people each is a rather challenging task.

Scorekeepers not only identify hits, errors, chances, runs, outs, assists, stolen bases, and many other "stats" each inning of a baseball game, but they also run cumulative totals of these for a final total. Then, other stat keepers enter these numbers into record books maintained on each player, team, division and league.

Without the least hesitation, a boy of ten can tell you his favorite baseball hero's batting average, how many hits the player had in last night's game, and the number of bases stolen. Not only that, he can probably tell you exactly when the hits were made and the bases stolen. More, he can give you the value of the feats in terms of home runs, singles, stealing third or taking home on a squeeze bunt.

Whether the individual is an actual player or a devout fan, the knowledge about scores and games remembered over years is quite incredible. Fans and players can recall certain instances as though they happened just a moment ago. A pitcher can tell you pitch-for-pitch how he struck out the side during a game played ten years ago. A fan who watched the game will recall other highlights of the same game with equal enthusiasm.

The human brain is amazing for its ability to store and recall such events and data. In a much simpler way, computers can be programmed to do the same sort of thing.

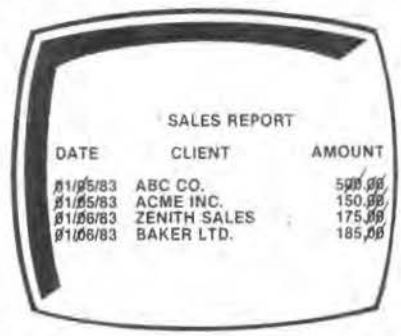
You will see how a computer can be programmed to provide intermediate totals so that you could, indeed, create a program to keep track of bowling scores or the feats of your favorite athlete. And the very same process used to maintain game scores by subtotal and final total can be applied to dozens of situations in business and industry. Let's see how to program them.

INTERMEDIATE TOTALS FOR A SALES PROGRAM

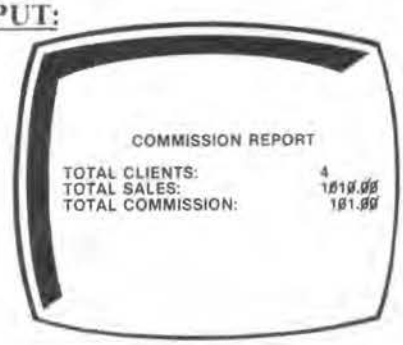
In many applications, input data is divided into groups, and intermediate totals are taken for each group. For example, suppose you are a salesperson working for a company on a commission basis of 1Ø%. It would be a relatively simple task for us to write a program which reads input data consisting of the date of the sale, the client's name, and the amount of the sale, and then to display on a screen the total commission owed to you. Such a program might look like this:

INPUT DATA:

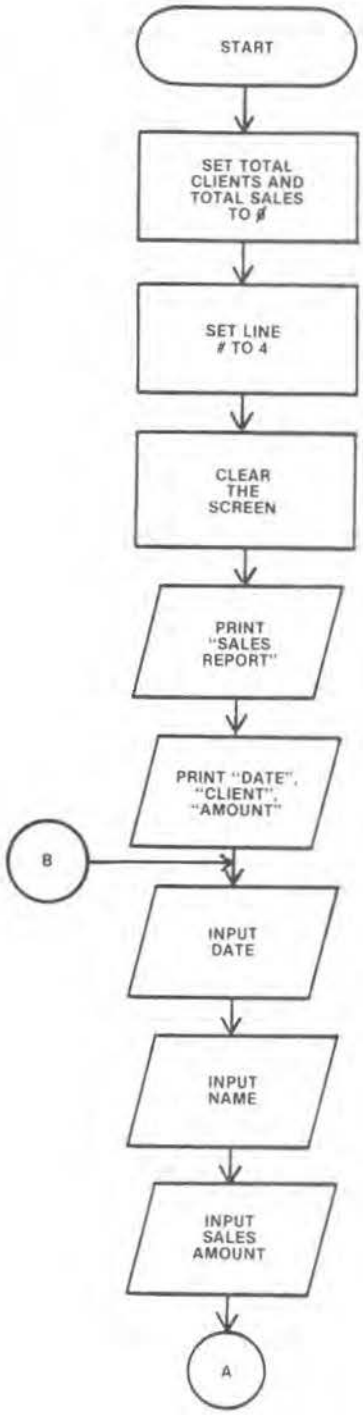
DATE OF SALE	CLIENT'S NAME	AMOUNT OF SALE
Ø1/Ø5/83	ABC CO.	5ØØ.ØØ
Ø1/Ø5/83	ACME INC.	15Ø.ØØ
Ø1/Ø6/83	ZENITH SALES	175.ØØ
Ø1/Ø6/83	BAKER LTD.	185.ØØ



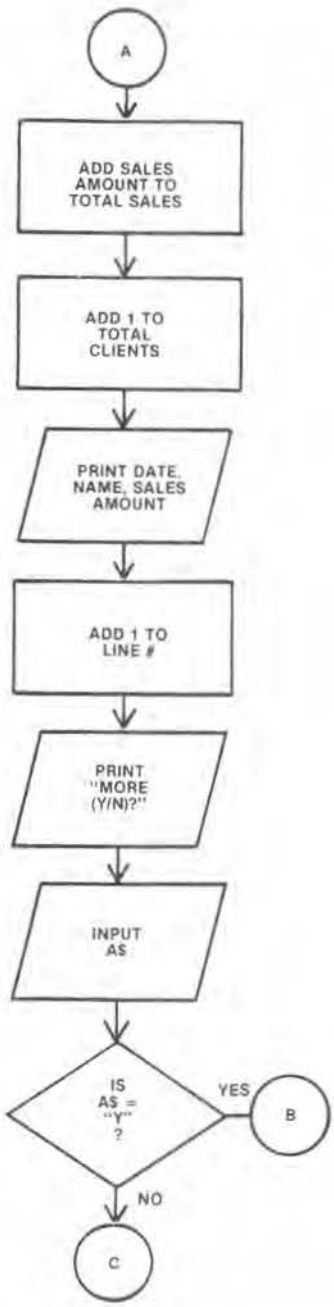
OUTPUT:



FLOWCHART




```
1Ø REM SALES COMMISSION
   REPORT — FINAL TOTALS ONLY
2Ø REM   VARIABLES    MEANINGS
3Ø REM   D$           DATE OF SALE
4Ø REM   N$           NAME OF CLIENT
5Ø REM   S            AMOUNT OF SALE
6Ø REM   T1           TOTAL SALES
7Ø REM   T2           TOTAL CLIENTS
8Ø REM   T3           TOTAL COMMISSION
85 REM   L            LINE NUMBER
9Ø REM   A$           PROMPT TO CONTINUE LOOP
1ØØ LET T1 = Ø
11Ø LET T2 = Ø
115 LET L = 4
12Ø CLS
13Ø PRINT TAB 9; "SALES REPORT"
14Ø PRINT TAB 2; "DATE"; TAB 1Ø;
   "CLIENT"; TAB 24; "AMOUNT"
145 PRINT AT 21,Ø; "ENTER DATE:"
15Ø INPUT D$
155 PRINT AT 21, Ø; "ENTER NAME:"
16Ø INPUT N$
165 PRINT AT 21,Ø; "ENTER SALES:"
17Ø INPUT S
```




```
18Ø LET T1 = T1 + S
19Ø LET T2 = T2 + 1
2ØØ PRINT AT L,Ø; D$; TAB 1Ø; N$;
    TAB 24; S
2Ø5 LET L = L + 1
21Ø PRINT AT 21,Ø; "MORE (Y/N) ? "
22Ø INPUT A$
23Ø IF A$ = "Y" THEN GOTO 145
```


```
24Ø LET T3 = T1 * .1Ø
25Ø CLS
26Ø PRINT TAB 6; "COMMISSION
    REPORT"
27Ø PRINT "TOTAL CLIENTS: ";
    TAB 2Ø; T2
28Ø PRINT "TOTAL SALES: ";
    TAB 2Ø; T1
29Ø PRINT "TOTAL COMMISSION: ";
    TAB 21; T3
3ØØ STOP
```

Enter this program and observe how it runs. Use the input data on Page 2.

The AT function can be used to control displayed PRINT lines. Like the TAB function, it causes data to be printed beginning in the specified horizontal column. In addition, it also specifies the vertical line number where the data should be printed. The format of the PRINT AT statement is:

nn PRINT AT L,C; constant or variable

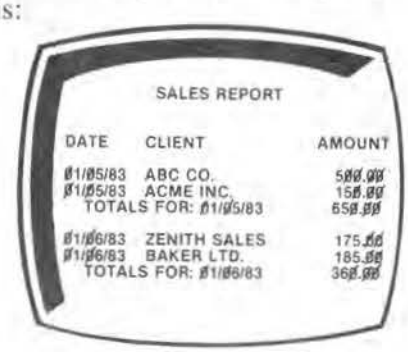
where: nn is the line number of the instruction
PRINT is a BASIC keyword
AT is a BASIC function
L is the line number (Ø is the first line on the screen and 21 is the last line)
C is the column number (Ø is the leftmost position and 31 is the rightmost)

Note that all prompts are printed at the bottom of the screen through the use of the PRINT AT 21,Ø statements (lines 145, 155, 165 and 21ØØ). This allows us to display the output of the sales report as we are entering new data. However, as soon as we wish to display one record's worth of output, our computer will try to print on the next line.

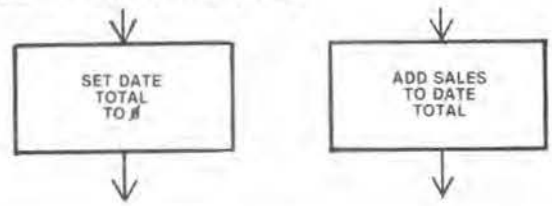
Since there is no next line to print on, statement number 2ØØ tells the computer to print on the "L" line number. This was initially set to 4 for the first detail record (statement number 115) and then incremented by 1 for the next record (line number 2Ø5). Otherwise, the program merely accepts the input and accumulates totals so that we can display the commission amount after all data has been entered.

But now, let's suppose we wished to have a breakdown of the totals entered for each day. In the above example, this would mean one subtotal for Ø1/Ø5/83 and another for Ø1/Ø6/83; that is, for each new date entered we want a total for the previous date. This is what we mean by control breaks and intermediate totals.

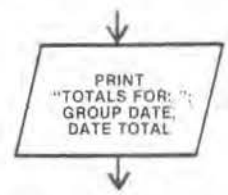
Let's now develop some logic which we can insert into our sample program to produce output like this:



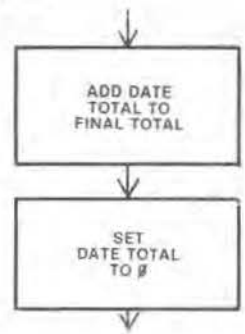
In order to add the two subtotal lines illustrated above, we must add several steps to our flowchart. First, we'll need another accumulator so that we will be able to display the total for each day. This accumulator will be initialized to zero at the start of the program. For each record, we'll add the sales amount to it.



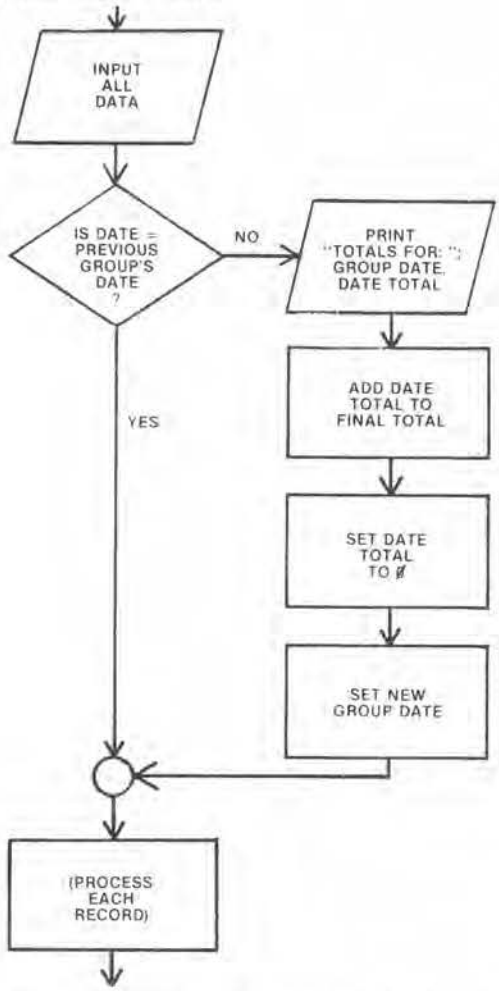
When the date entered changes, a line will be printed containing this total and the date to which it applies.



Then we'll add this total to our final total accumulator. The date total accumulator must be reset to zero, so that we can accumulate a total for the next date.



Next, we need to add some logic to test each date entered to see if it is different from the previous group's date. If it is, we'll do our subtotal logic and then rejoin the logic required to process each record.



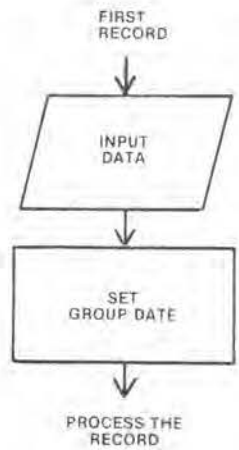
Note that we will have to establish another variable in our program—the group date. This date identifies our control groups and, thus, is known as the control field. It will be set for each group of sale dates and will be compared to each time we enter a new date.

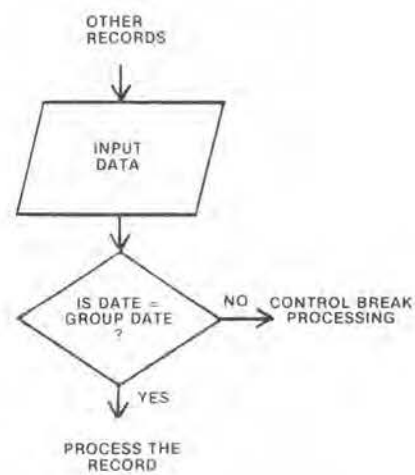
There is an assumption that must be made whenever control-break logic is applied. The data being entered must be grouped or sorted prior to becoming input to a program. In the data we are using, all records with the same date are entered consecutively. Our logic, then, is able to state that whenever a new date is entered, *all* records for the previous group's date have already been processed.

If another record with a previous group's date is entered, our program will treat it as another group entirely.

There are two problems which control-break logic must solve, and they deal with the first and last control groups.

We can't process the first record the same way as the rest of the records, since we won't have any previous group date to compare it with. What we have to do is to take the first date entered and establish it as the first group date. Every other date entered will have to be compared against the first group date. Here are the two sets of logic that will be necessary.

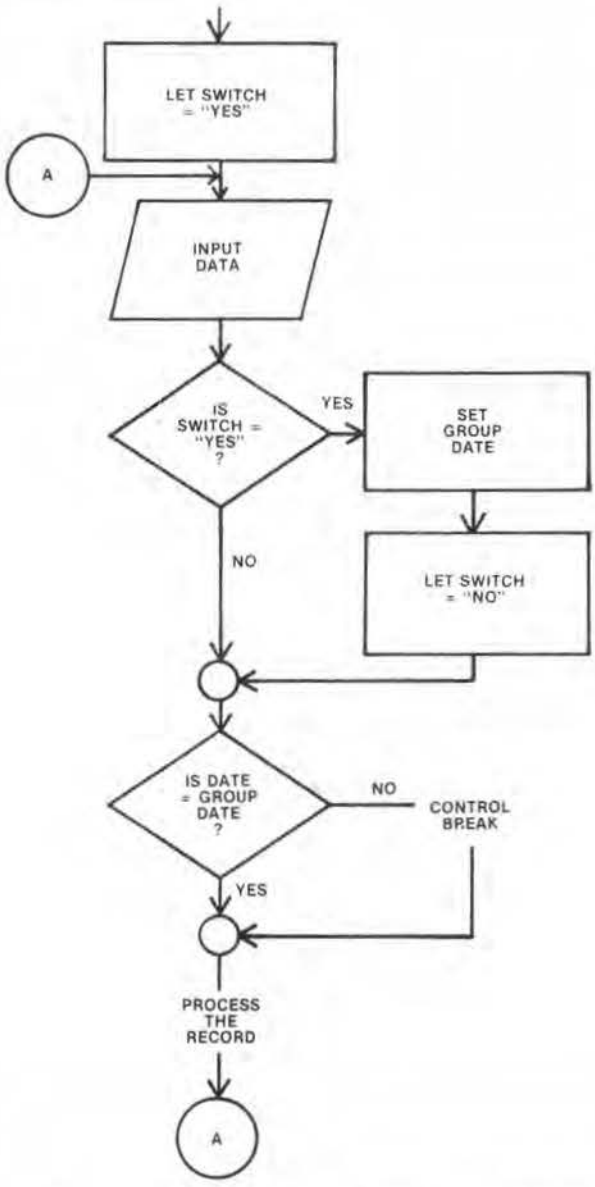




Other than the fact that the first record will cause a "false" control break, the rest of the processing is the same for all records.

One way to resolve this problem is by establishing a "switch." A switch is a programmer-defined variable which controls the logical path a record will follow, much as a railroad switch controls the path a train will take. Program switches (also known as indicators or flags) take on one of two possible values: YES or NO, ON or OFF, 1 or Ø, etc.

Observe how the use of a switch solves our problem:

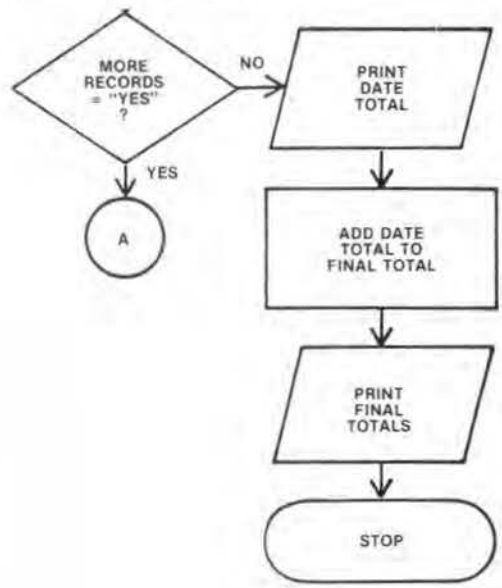


In this flowchart, a switch is defined and initialized to a value of "YES." After the first date is entered, a test is made to see if the switch is equal to "YES," which, of course, it is. A branch to the right occurs which sets the group date and then sets the switch to "NO." When the main logic is rejoined, no control break is found and the record is processed. When the next record is submitted as input, the switch no longer equals "YES" and so a control break will occur if we have entered a new date.

Rarely is the need for a switch obvious in the initial analysis of a program. Rather, programmers will set up a switch when alternate paths in a program must be taken (although there is nothing on the record itself which will decide which path to take). Switches are useful in a variety of applications. They allow a record to follow the correct path, depending on the path that a previous record took.

Now that we have solved the problem of the first group date, how about the situation the last record causes. How will the computer know when to display the last group totals? For all other records, we know only that we are to print a date total line *after* we have entered a record with a different date. When we enter an "N" to discontinue our loop, our last group's total will not have been processed.

This problem is more easily solved than the first record problem. We will be correct if we assume that the last group total should always be printed immediately before the final totals are taken as in the flowchart below.



Note that the steps "PRINT DATE TOTAL" and "ADD DATE TOTAL TO FINAL TOTAL" are the same steps as we have used in our regular control-break logic. The only difference is the next step. In our control-break logic, our next step is to reset the date total to zero. In this final total logic, the next step is to display the final totals.

We could code these steps twice in our program, but it would be preferable to branch to these two steps and then to go back. The only problem is, to what do we go back? A GOTO statement will get us to these instructions, but will not be able to get us back.

A SUBROUTINE

A subroutine is a set of instructions which can be executed from anywhere in the program and will then return to the statement which "called" for it. In a flowchart, a subroutine is coded as a processing box with two parallel lines inside of it as in:

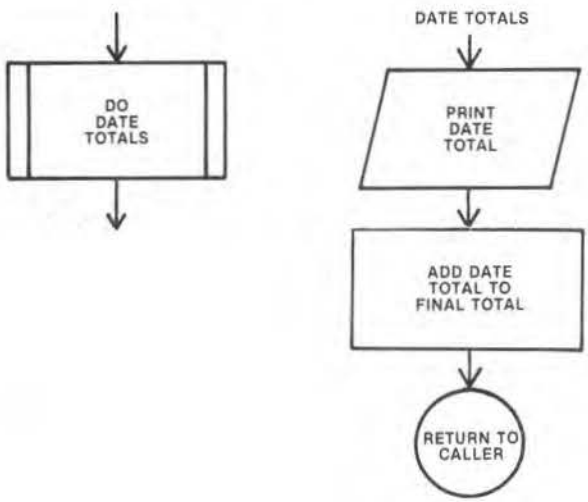


FIGURE 2—The programmer spends a certain amount of time with clients to learn exact specifications and applications. Then, it is back to the drawing board and the computer.

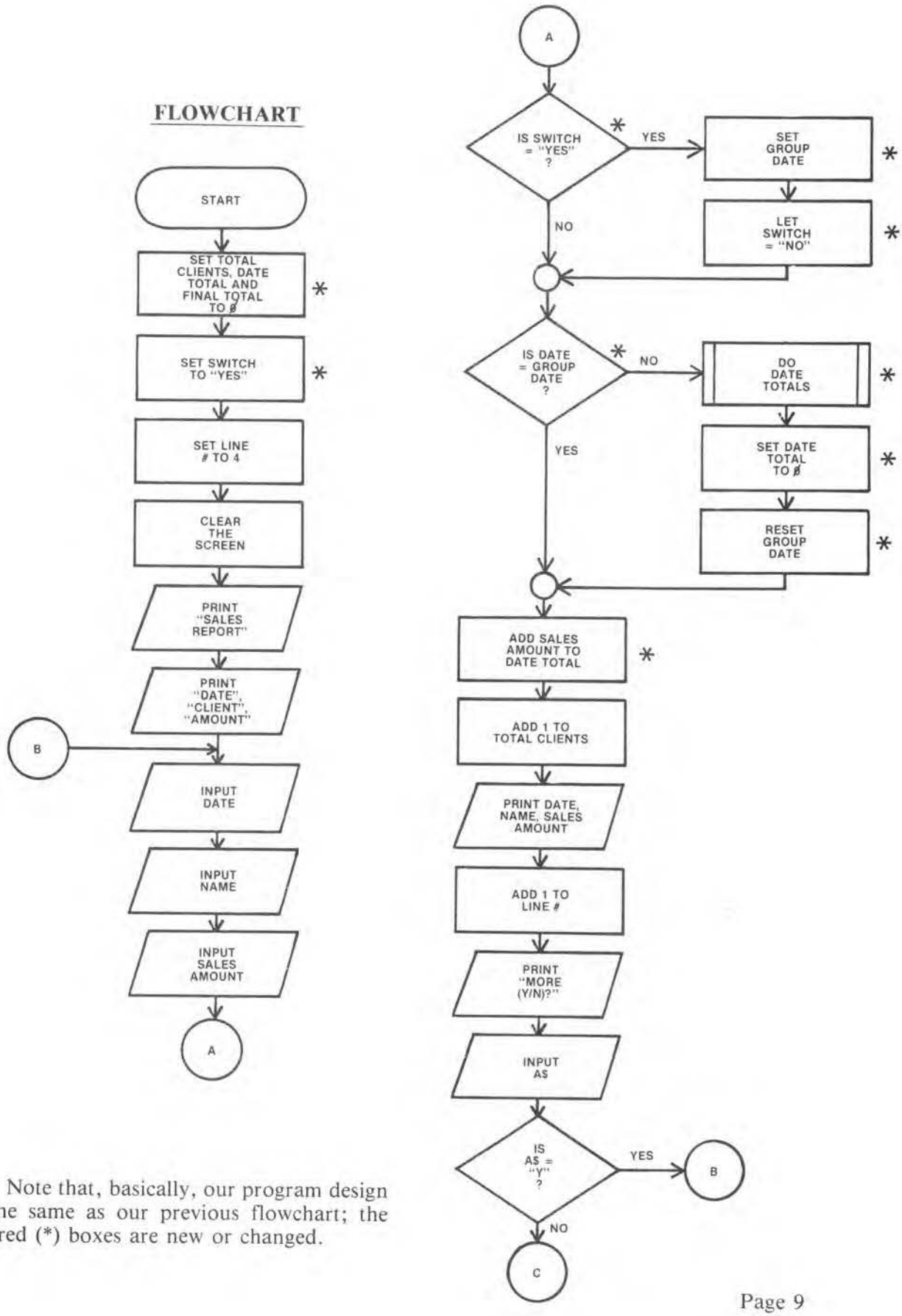
When the step "DO DATE TOTALS" is encountered, a branch will occur to print the date totals and add to the final total. After these steps are done, the "RETURN TO CALLER" instruction will cause a branch back to the statement "DO DATE TOTALS" so that it can do the next step.

In our program, we will do the date totals from two places: when a control break occurs and before we print the final totals.

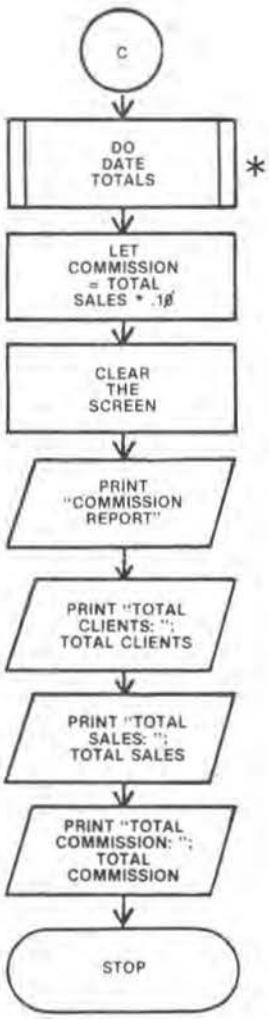
NOTE: A subroutine causes a branch and a return; a "GOTO" causes a branch with no return.

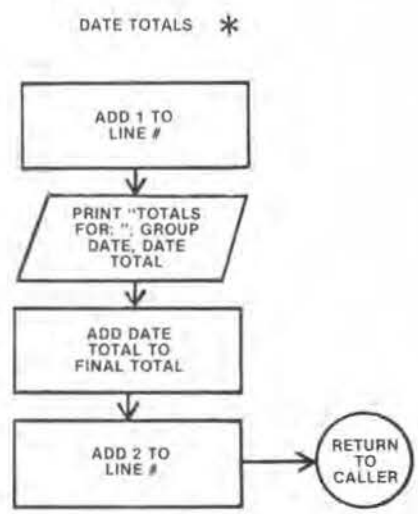
Let us now insert control-break logic into our existing sales report program. The entire flowchart follows:

FLOWCHART



Note that, basically, our program design is the same as our previous flowchart; the starred (*) boxes are new or changed.





Now let's make these changes to our old BASIC program. If we wish to fully maintain our remarks section, we will need to describe the additional variables used by our program:

```
82 REM T4 DATE GROUP TOTAL

92 REM F$ FIRST RECORD SWITCH

93 REM G$ GROUP DATE
```

To our program logic, add a step to set the date total accumulator to Ø:

```
112 LET T4 = Ø
```

We next have to initialize our switch (F$) to a value of "YES." Add this statement:

```
114  LET F$ = "YES"
```

The next instruction to insert is the one which will test our switch within the loop to see whether it is the first date entered. Add:

```
171 IF F$<>"YES" THEN GOTO 174

172 LET G$ = D$

173  LET F$ = "NO"
```

Now, if the switch is equal to "YES," steps 172 and 173 will be executed. 172 will set the group date equal to the date just entered. Statement 173 will set the switch to "NO" so that all subsequent loops through our program will cause steps 172 and 173 to be bypassed (since we never set the switch back to "YES").

Our next group of steps will test for a change in the date entered:

```
174 IF D$ = G$ THEN GOTO 18Ø

175 GOSUB 31Ø

177 LET T4 = Ø

178 LET G$ = D$
```

Step 174 checks for a control break; if none has occurred, we just process the record as before. If a control break happens, though, statement 175 "GOSUB 31Ø" says: branch to instruction 31Ø (where the date total statements begin).

GOSUB nnn

where nnn is a line number

After returning, set the date total accumulator back to Ø (177) and reset the group date comparing variable (178).

We must now change line 18Ø so that we are adding the sales amount to the date total accumulator rather than to the final total accumulator:

```
18Ø LET T4 = T4 + S
```

Now, we can skip to our final total routine and insert the instruction which will cause the last date totals to be displayed. Add:

```
235 GOSUB 31Ø
```

All that remains is to code the subroutine for the date totals to the end of our program:

```
3Ø5 REM DATE TOTALS BEGIN HERE

31Ø LET L = L + 1

32Ø PRINT AT L, Ø; "TOTALS FOR: ";
    G$; TAB 24; T4

33Ø LET T1 = T1 + T4

34Ø LET L = L + 2

35Ø RETURN
```

A remarks line (3Ø5) should be added to the beginning of our subroutine to ease readability and debugging. Lines 31Ø and 34Ø will format our output so that we will have a blank line before and after our date total line. Statement 32Ø prints the group date and the date total accumulator. At line 33Ø, we will be adding each date total to the final total.

ANTICIPATING PROGRAM CHANGES

```
174 IF D$ = G$ THEN GOTO 18Ø

175 GOSUB 31Ø

177 LET T4 = Ø

178 LET G$ = D$
```

FIGURE 3—When adding lines to a program, always try to leave "extra" lines open for more changes. In this instance, line #176 has not been used so it will remain available.

Line 35Ø, RETURN, is crucial to our program. It will cause a branch back to the instruction which called for the date totals to be done (either line 175 or line 235) so that the next instructions will be executed correctly.

Your final program should now look like this:

```
1ØØ LET T1 = Ø

11Ø LET T2 = Ø

112 LET T4 = Ø

114 LET F$ = "YES"

115 LET L = 4

12Ø CLS

13Ø PRINT TAB 9; "SALES REPORT"

14Ø PRINT TAB 2; "DATE"; TAB 1Ø;
    "CLIENT"; TAB 24; "AMOUNT"

145 PRINT AT 21,Ø; "ENTER DATE:"

15Ø INPUT D$

155 PRINT AT 21,Ø; "ENTER NAME:"

16Ø INPUT N$

165 PRINT AT 21,Ø; "ENTER SALES:"
```

```
17Ø INPUT S
171 IF F$<>"YES" THEN GOTO 174
172 LET G$ = D$
173 LET F$ = "NO"
174 IF D$ = G$ THEN GOTO 18Ø
175 GOSUB 31Ø
177 LET T4 = Ø
178 LET G$ = D$
18Ø LET T4 = T4 + S
19Ø LET T2 = T2 + 1
2ØØ PRINT AT L, Ø; D$; TAB 1Ø; N$;
    TAB 24; S
2Ø5 LET L = L + 1
21Ø PRINT AT 21,Ø ; "MORE (Y/N) ? "
22Ø INPUT A$
23Ø IF A$ = "Y" THEN GOTO 145
235 GOSUB 31Ø
237 PAUSE 32767
24Ø LET T3 = T1 * .1Ø
25Ø CLS
26Ø PRINT TAB 6; "COMMISSION
    REPORT"
27Ø PRINT "TOTAL CLIENTS: "; TAB
    2Ø; T2
28Ø PRINT "TOTAL SALES: ";
    TAB 2Ø; T1
29Ø PRINT "TOTAL COMMISSION: ";
    TAB 21; T3
3ØØ STOP
3Ø5 REM DATE TOTALS BEGIN HERE
31Ø LET L = L + 1
32Ø PRINT AT L, Ø; "TOTALS FOR: ";
    G$; TAB 24; T4
33Ø LET T1 = T1 + T4
34Ø LET L = L + 2
35Ø RETURN
```

Run this program with the exact same data we used for the program at the beginning of this Unit. See how the only changes are the date totals which are displayed for Ø1/Ø5/83 and Ø1/Ø6/83.

Before we walk through this program to see exactly how it works, you may perhaps have noticed a slight problem. That is, when we respond with an "N" to stop our loop, the last date totals briefly appear on the screen, then quickly disappear when the final totals are displayed.

FIGURE 4—Debugging sometimes requires long hours of checking flowcharts and entering data step-by-step, over and over again. When it gets too frustrating, take a break! You deserve it.

CREATING A PROGRAM PAUSE

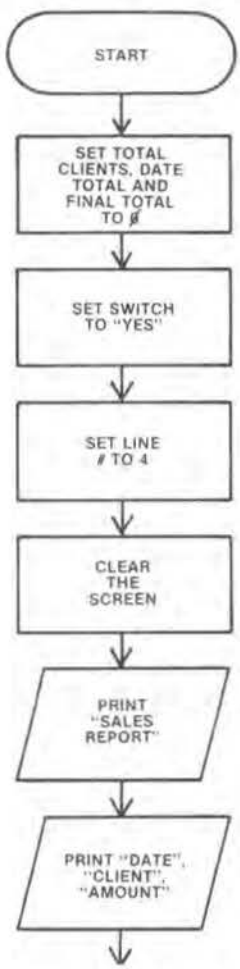
We can have the program pause at this point before the final totals are printed by inserting one more statement:

237 PAUSE n
where n is any number

When encountered, this statement will halt execution until **any key on the keyboard is pressed**. Insert this line with a value of "32767" and run the program one more time. See how it works?

A value of 32767 will cause the pause to last for about 1Ø minutes. A lesser value will cause the pause to end either when a key is pressed or at an amount of time equal to:

$\frac{n}{5Ø}$ = number of seconds PAUSE will last if no key is pressed.



The PAUSE statement can serve many useful functions. It is most often used to allow the user to read the output on the screen at his or her own pace. The screen will only be cleared when a key is pressed (other than the BREAK key). Then the program will continue by executing the next statement.

Since the duration of a PAUSE statement can be carefully controlled, it can also be used to function as a clock; i.e. PAUSE 5Ø would last for one second, while PAUSE 3ØØØ for about one minute. Adjusting the number of the PAUSE may be necessary depending on the degree of accuracy desired.

Remember, the number of the PAUSE controls the length of the PAUSE if *no* key is pressed. Pressing a key will always terminate the PAUSE immediately, regardless of the number.

INITIALIZATION ROUTINE

```
1ØØ LET T1 = Ø

11Ø LET T2 = Ø

112 LET T4 = Ø

114 LET F$ = "YES"

115 LET L = 4

12Ø CLS

13Ø PRINT TAB 9;
    "SALES REPORT"

14Ø PRINT TAB 2;
    "DATE"; TAB 1Ø;
    "CLIENT"; TAB 24;
    "AMOUNT"
```

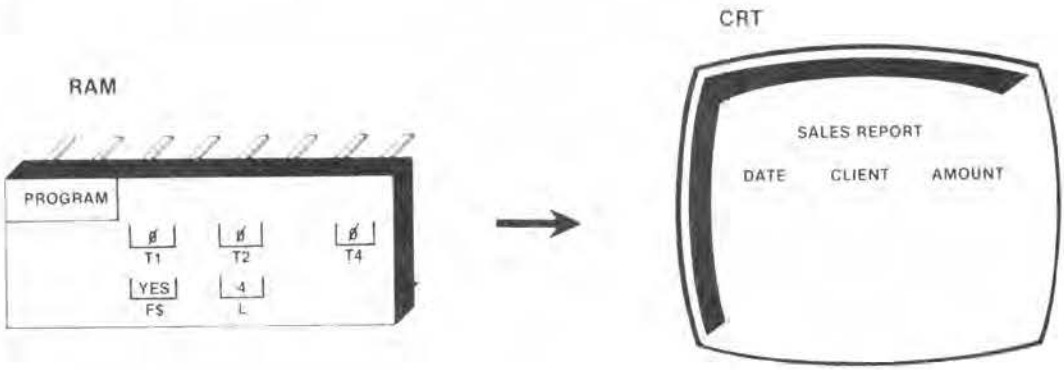
SAMPLE PROGRAM WALK THROUGH

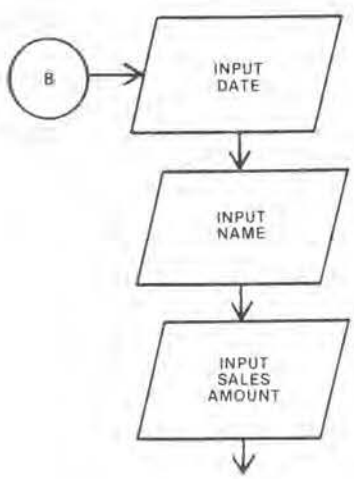
As we walk through this program, let's pay special attention to these four sets of instructions:

1. The first record switch instructions.
2. The control field and control breaks.
3. The GOSUB and RETURN statements.
4. The line number statements.

Our first three statements set our accumulators to zero; the fourth initializes the value of our switch to YES, and the fifth sets the line number variable to 4. Next, we display our headings after clearing the screen.

Prior to entering our loop, RAM and the CRT look like this:





INPUT DATA

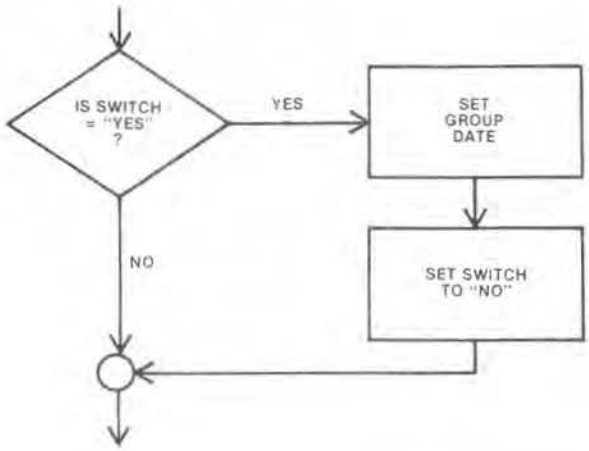
```
145 PRINT AT 21, Ø;
    "ENTER DATE:"

15Ø INPUT D$

155 PRINT AT 21,Ø;
    "ENTER NAME:"

16Ø INPUT N$

165 PRINT AT 21,Ø;
    "ENTER SALES:"

17Ø INPUT S
```

These six instructions cause prompts for three variables to be entered by the operator. Let's assume that we've submitted the data from our first record:

Ø1/Ø5/83 ABC CO. 5ØØ.ØØ

FIRST RECORD TEST

Before checking for a control break, we now test to see if this was the first record.



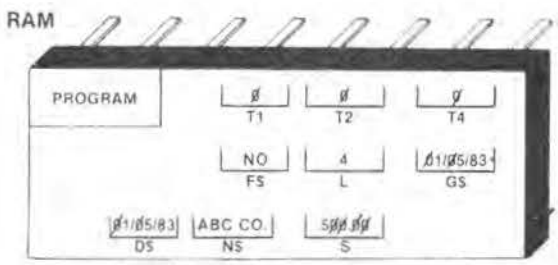
171 IF F$ <> "YES" THEN GOTO 174

172 LET G$ = D$

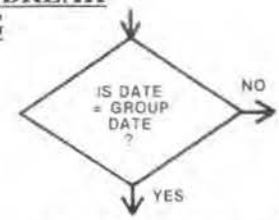
173 LET F$ = "NO"

The statements, in essence, say, if the variable F$ is not equal to "YES," then we can bypass the next two instructions. At this point in our walk through, the switch does equal "YES" (having been set that way in the initialization routine), so our date comparison area (G$) is given the value of the first date entered (D$). Next, we set the switch to "NO" ensuring that in subsequent loops through the program, statements 172 and 173 will never again be executed. (There is only one first record per RUN!)

Our computer storage now looks this way:



CONTROL BREAK CHECKING

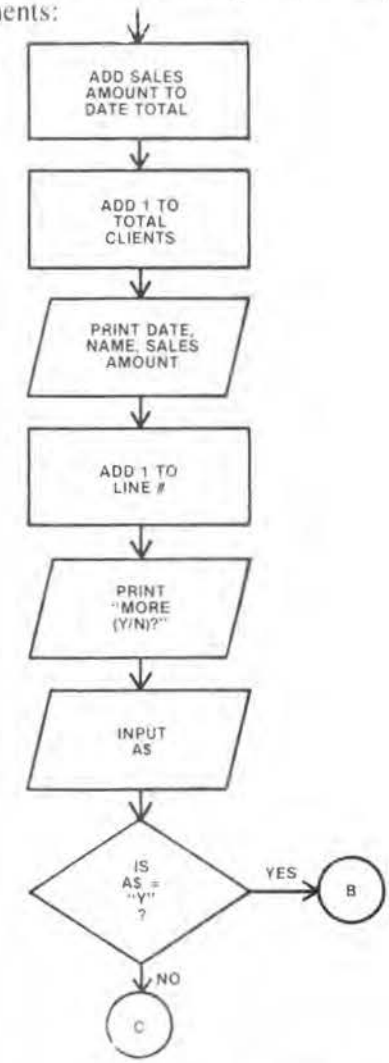


174 IF D$ = G$ THEN GOTO 18Ø

This comparison tests to see if a new date was just entered. If so, a control break will occur—but not a "false" control break! This is because we have just forced G$ to be the same as D$ in our first record routine. Our logic will take the "YES" path.

DETAIL PROCESSING

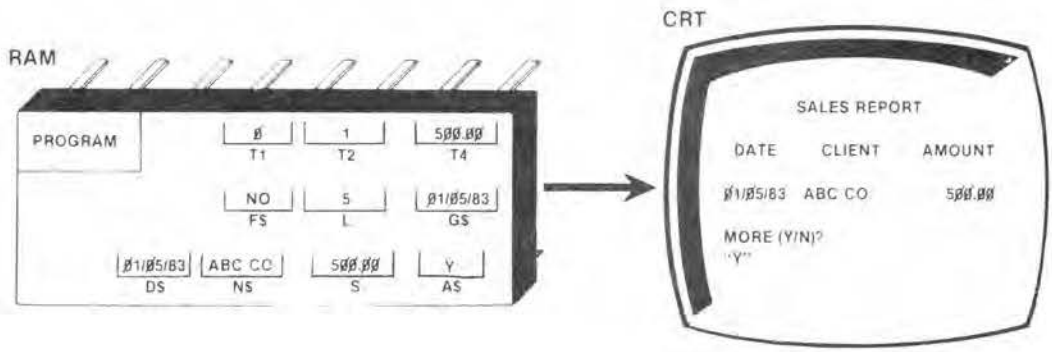
Every record will eventually find its way to these statements:



```
18Ø LET T4 = T4 + S
19Ø LET T2 = T2 + 1
2ØØ PRINT AT L, Ø; D$; TAB 1Ø; N$;
    TAB 24; S
2Ø5 LET L = L + 1
21Ø PRINT AT 21,Ø; "MORE (Y/N) ? "
22Ø INPUT A$
23Ø IF A$ = "Y" THEN GOTO 145
```

These instructions should seem fairly familiar to us by now. They cause the accumulation of the date total sales and of the number of records entered. Note that the printing of our detail line will occur at the line number specified by the value of L which will then be incremented by one for the next line.

A prompt is then issued to continue the loop to which we respond with "Y." Just before our loop is repeated, our system has this appearance:

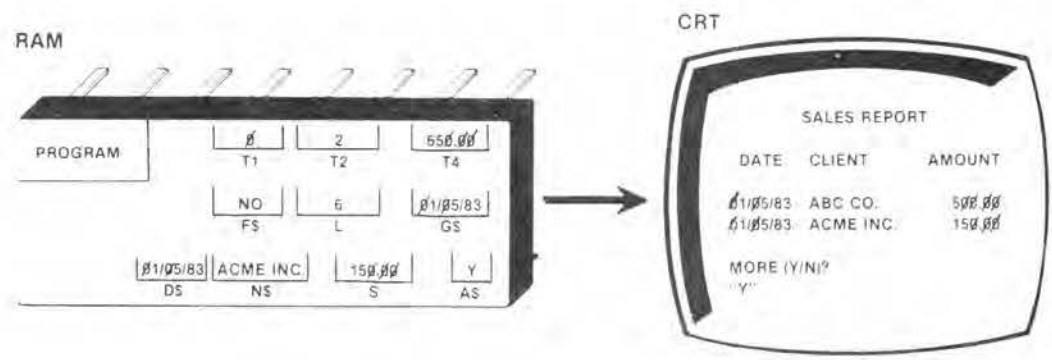


SECOND PASS THROUGH LOOP

For the next record, we'll enter the following values:

Ø1/Ø5/83 ACME INC. 15Ø.ØØ

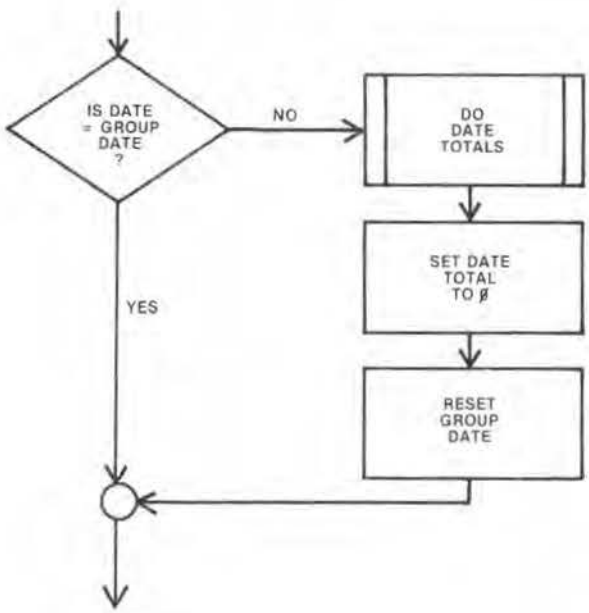
This time, those statements which were set up for the first record will *not* be done because F$ (our switch) now has a value of "NO." Neither will a control break occur, since this date is identical to the group date. Therefore, this record will be processed similarly to the first, and our system now looks like this:



THIRD PASS

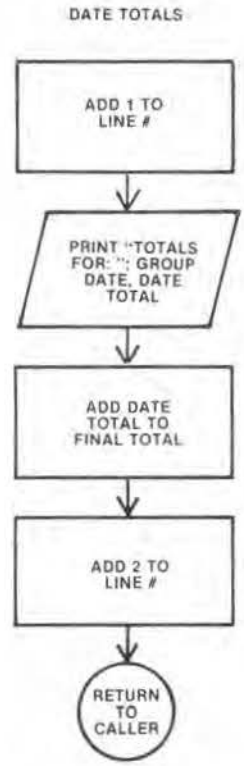
If we were processing this program manually, we would know to write the date total line since we have printed all Ø1/Ø5/83 records already. But our computer cannot project into the future; it doesn't know that it already has processed a group of records until a new date is entered.

After we have entered our third record (Ø1/Ø6/83 ZENITH SALES 175.ØØ), a control break will occur since this date, D$, is not equal to the previous group date, G$.




```
174 IF D$ = G$ THEN GOTO 18Ø
175 GOSUB 31Ø
177 LET T4 = Ø
178 LET G$ = D$
```

Statement 175 causes a branch to line 31Ø; since it's a GOSUB. However, a RETURN instruction will cause a branch back to this statement. Our date total subroutine is now executed:



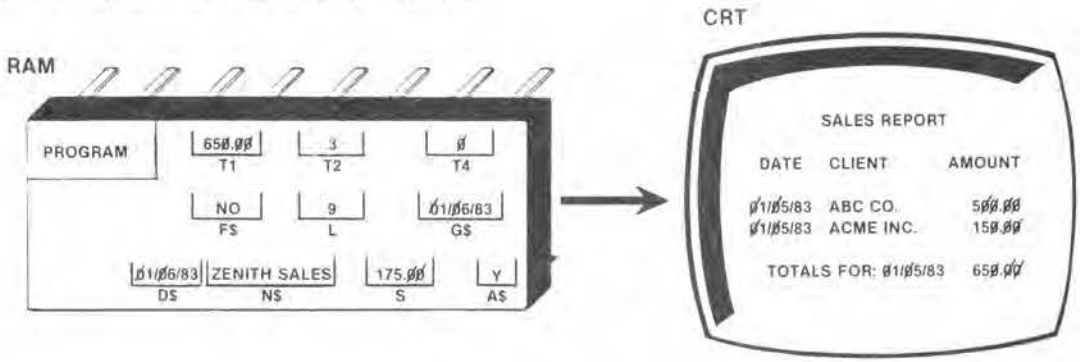

```
3Ø5 REM DATE TOTALS BEGIN HERE
31Ø LET L = L + 1
32Ø PRINT AT L, Ø; "TOTALS FOR: "; G$;
    TAB 24; T4
33Ø LET T1 = T1 + T4
34Ø LET L = L + 2
35Ø RETURN
```

When the GOSUB on statement 175 is executed, an area of main storage will hold statement number "175" so that, when the return is encountered, the proper branch-back will occur. This area is known as the "RETURN STACK."

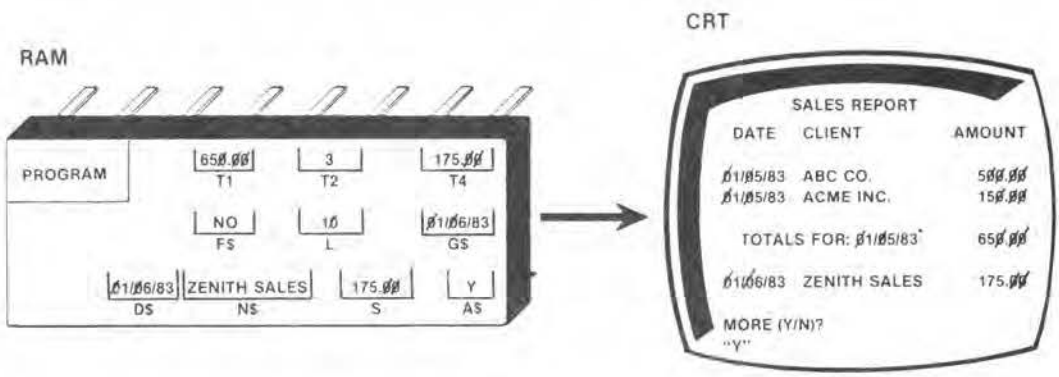
The first statement in the subroutine increments the line counter by one so that statement 32Ø will print two lines below the last detail line. Note: The date printed on this line is the "group" date of the previous record and *not* the date just entered (D$).

Instruction 33Ø adds the group total accumulator to the final total accumulator. Next, the line counter is upped by two, so that the following detail line will be double spaced.

The RETURN statement on line 35Ø causes the control unit to fetch the instruction located at the address in the return stack (line 175) and to execute the next statement. This step (line 177) sets the group total back to Ø and then the group date is set to the date which caused the control break. Just before rejoining our detail processing logic, here is what we have:



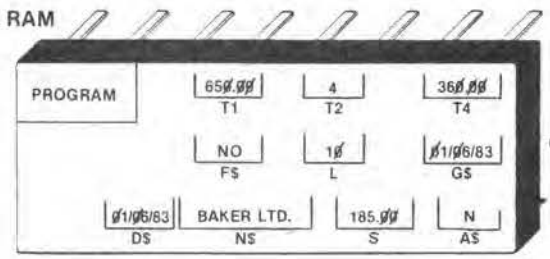
The detail processing lines 18Ø through 23Ø cause a line of output to appear two spaces below the group total line, giving us this:



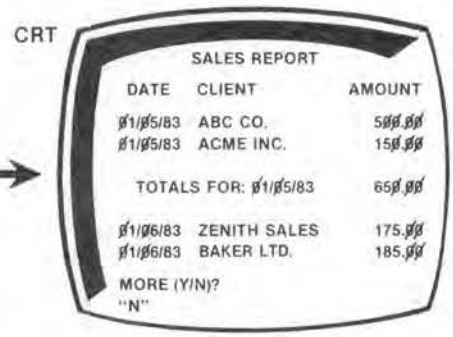
FOURTH PASS

In our sample data, we have one more record to process:

Ø1/Ø6/83 BAKER LTD. 185.ØØ

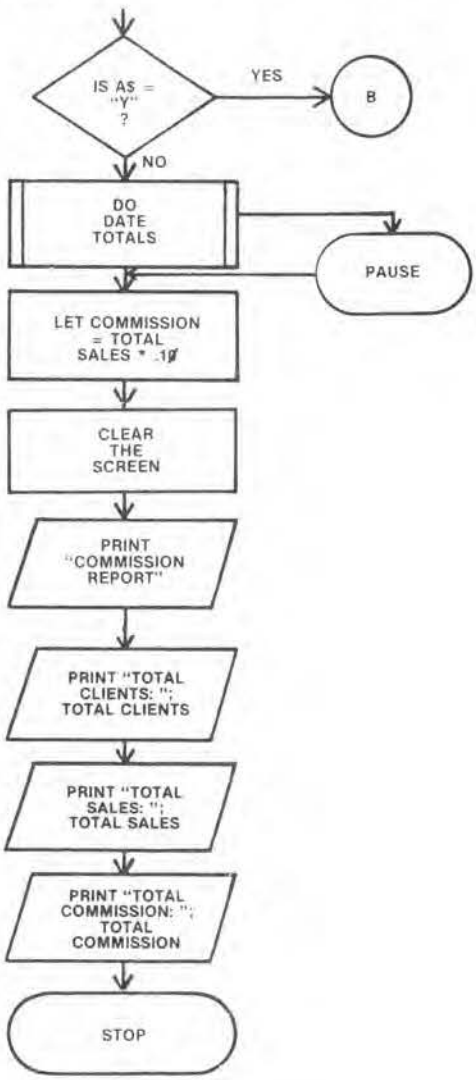


Since this record will not cause a control break, only detail processing will occur. Another line will be displayed and totals accumulated to produce the following:



END OF FILE PROCESSING

Now that we have entered all of our records, a response of "N" to "MORE (Y/N)?" should be entered. Although the program logic leads to the printing of final totals, the last control group must be processed before this can happen. Remember, the last group totals are always "forced" at the end of the file.

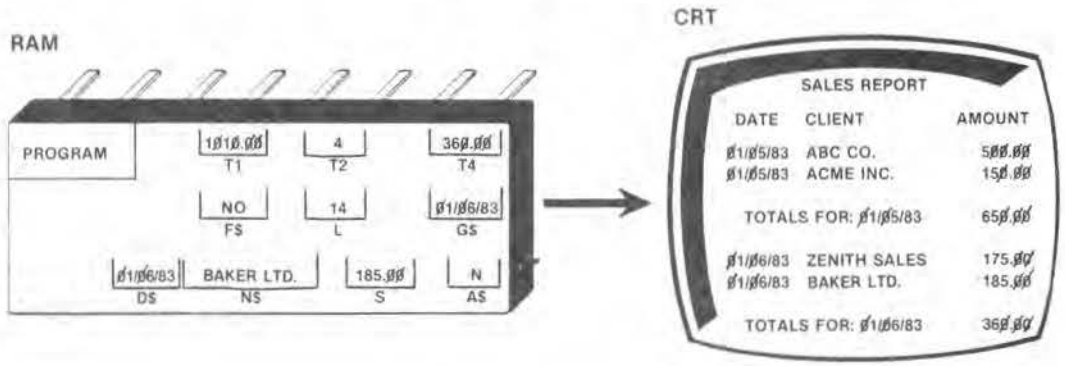



```
23Ø IF A$ = "Y" THEN GOTO 145
235 GOSUB 31Ø
237 PAUSE 32767
24Ø LET T3 = T1 * .1Ø
25Ø CLS
26Ø PRINT TAB 6; "COMMISSION
    REPORT"
27Ø PRINT "TOTAL CLIENTS: ";
    TAB 2Ø; T2
28Ø PRINT "TOTAL SALES: ";
    TAB 2Ø; T1
29Ø PRINT "TOTAL COMMISSION: ";
    TAB 21; T3
3ØØ STOP
```

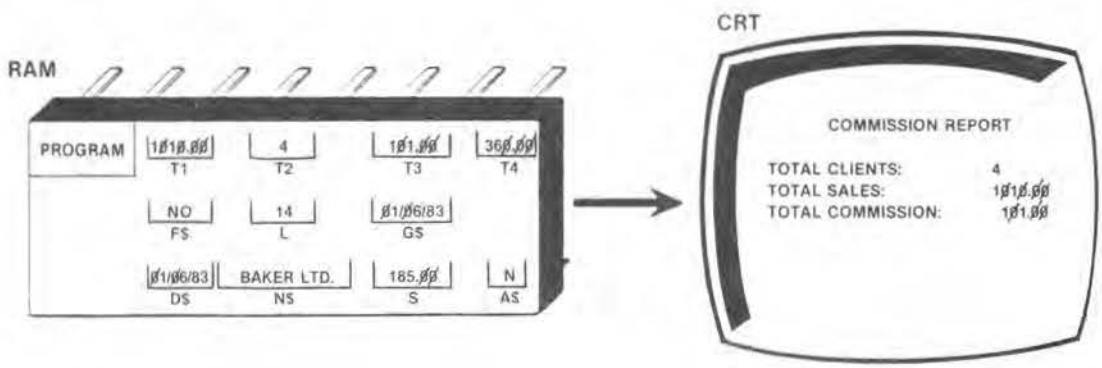
Statement 235 says, "Go do the subroutine beginning at line 31Ø and come back." Group totals for Ø1/Ø6/83 will be displayed and final totals accumulated. The RETURN statement will cause a branch back to the caller. The PAUSE statement at line 237 will halt execution until a key is pressed.

The commission will be calculated at 1Ø% of the total sales; the screen is cleared and all totals displayed before the program ends. When the STOP statement is encountered, our system will look like this:

BEFORE PAUSE



AFTER PAUSE



GROUP PRINTING

Before we conclude this section, let's modify our program by deleting a few lines in order to illustrate group printing.

Delete lines 14Ø, 2ØØ and 2Ø5. These lines are responsible for displaying column headings and detail lines only. RUN the program using the same input data as we used previously. Note the changes in our output.

Group printing refers to the displaying of total lines only, also known as a summary report. Group printing allows the user to receive summarized data without becoming "bogged down" with too many details.

Whether or not a group printing is desired, the essential logic of control-break processing remains unchanged.

Now, to see what you've learned in this section, complete the Programmer's Check which follows.

PROGRAMMER'S CHECK

1

Designing Programs
With Control Breaks

Below is Program IU8A1. The specifications will challenge your ability as a programmer. Refer back to the text for similar line items and flowcharting if you have difficulty.

Program Name: **IU8A1**
Type: **CONTROL BREAKS**
Specifications:

Design and code a control-break program which will create a weekly payroll report. Input to this program will be TIME CARDS formated as:

INPUT
TIME CARDS

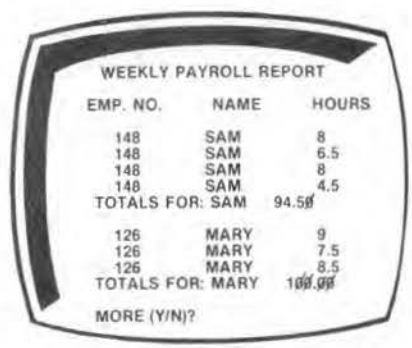
EMP. #	EMPLOYEE NAME	HOURS	RATE

TEST DATA:

148	SAM	8	3.5Ø
148	SAM	6.5	3.5Ø
148	SAM	8	3.5Ø
148	SAM	4.5	3.5Ø
126	MARY	9	4.ØØ
126	MARY	7.5	4.ØØ
126	MARY	8.5	4.ØØ

Output will consist of two screens:

DETAIL SCREEN

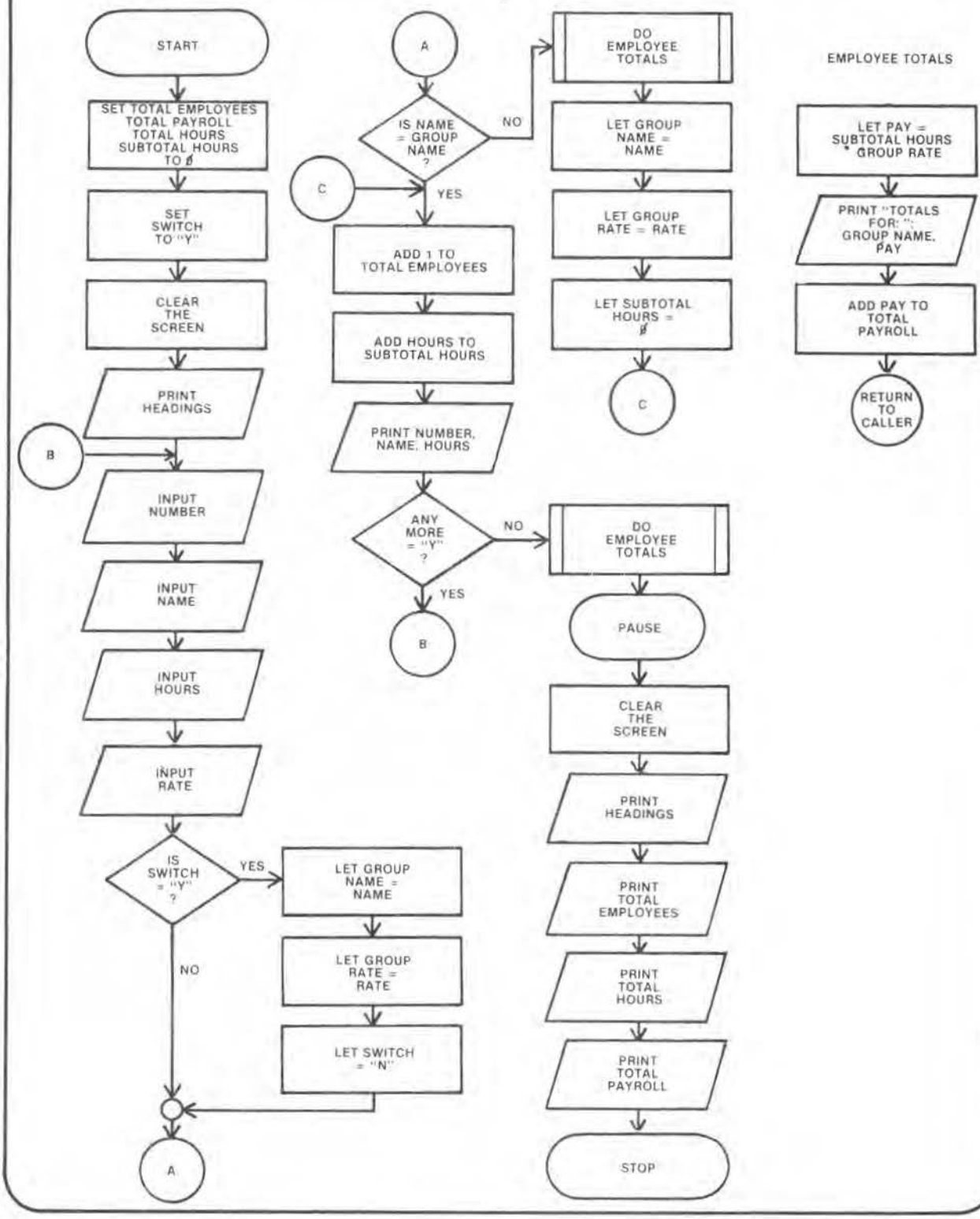


FINAL TOTAL SCREEN

(Answers on Pages 22 and 23)

PROGRAMMER'S CHECK ANSWERS

FLOWCHART SOLUTION TO IU8A1



Program Solution to IU8A1

```
10 REM IU8A1 WEEKLY
   PAYROLL PROGRAM
20 REM YOUR NAME—02/05/83
30 REM E          EMPLOYEE NUMBER
40 REM N$         EMPLOYEE NAME
50 REM H          HOURS
60 REM R          RATE
70 REM T1         TOTAL EMPLOYEES
80 REM T2         TOTAL PAYROLL
90 REM T3         SUBTOTAL HOURS
100 REM T4        TOTAL HOURS
110 REM P         EMPLOYEE'S PAY
120 REM G$        GROUP NAME
130 REM G1        GROUP RATE
140 REM S$        FIRST RECORD SWITCH
150 REM I         LINE COUNTER
160 REM A$        RESPONSE TO PROMPT
170 LET T1 = 0
180 LET T2 = 0
190 LET T3 = 0
195 LET T4 = 0
200 LET S$ = "Y"
210 LET L = 4
220 CLS
230 PRINT TAB 2; "WEEKLY
    PAYROLL REPORT"
240 PRINT "EMP. NO."; TAB 10; "NAME";
    TAB 20; "HOURS"
250 PRINT AT 21, 0; "ENTER NUMBER:"
260 INPUT E
270 PRINT AT 21, 0; "ENTER NAME : "
280 INPUT N$
290 PRINT AT 21, 0; "ENTER HOURS :"
300 INPUT H
310 PRINT AT 21,0; "ENTER RATE : "
320 INPUT R
330 IF S$< >"Y" THEN GOTO 370
340 LET G$ = N$
350 LET G1 = R
360 LET S$ = "N"
370 IF N$ = G$ THEN GOTO 430
380 GOSUB 580
390 LET G$ = N$
400 LET G1 = R
410 LET T3 = 0
430 LET T3 = T3 + H
440 PRINT AT L,0; E; TAB 10; N$;
    TAB 21; H
450 LET L = L + 1
460 PRINT AT 21,0; "MORE (Y/N) ? "
470 INPUT A$
480 IF A$ = "Y" THEN GOTO 250
490 GOSUB 580
500 PAUSE 32767
510 CLS
520 PRINT TAB 10; "PAYROLL
    TOTALS"
530 PRINT "NO. OF
    EMPLOYEES: "; TAB 24; T1
540 PRINT "TOTAL HOURS
    WORKED: "; TAB 24; T4
550 PRINT "TOTAL PAYROLL: ";
    TAB 24; T2
560 STOP
570 REM EMPLOYEE TOTAL
    ROUTINE
580 LET P = T3 * G1
585 LET T1 = T1 + 1
590 LET L = L + 2
600 PRINT AT L,1; "TOTALS
    FOR: "; G$; TAB 20; P
610 LET T2 = T2 + P
620 LET T4 = T4 + T3
630 LET L = L + 2
640 RETURN
```

MULTIPLE CONTROL BREAKS

Once you have mastered the processing of control breaks, you may find that, in some applications, even the groups of records must be further divided into subgroups. When this is necessary, multiple control-break logic will be employed.

For example, let's suppose that in the sample we saw in the previous section of this Unit, individual items were sold to each company. Modifying our input data, we could now have:

INPUT FORMAT

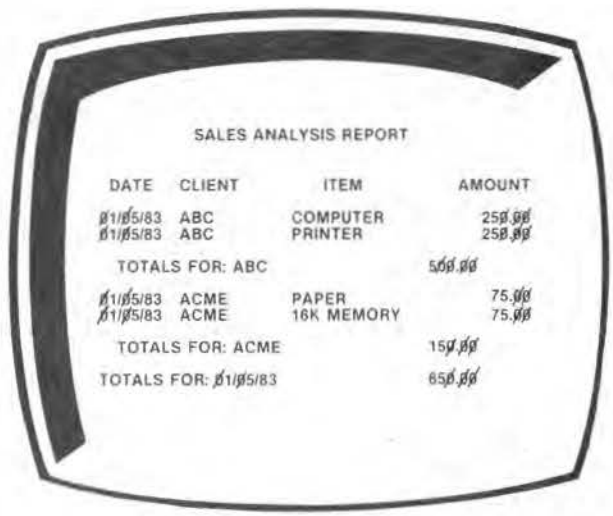
DATE	CLIENT	ITEM	AMOUNT
Ø1/Ø5/83	ABC	COMPUTER	25Ø.ØØ
Ø1/Ø5/83	ABC	PRINTER	25Ø.ØØ
Ø1/Ø5/83	ACME	PAPER	75.ØØ
Ø1/Ø5/83	ACME	16K MEMORY	75.ØØ
Ø1/Ø6/83	ZENITH	TAPE DRIVE	1ØØ.ØØ
Ø1/Ø6/83	ZENITH	DISKETTES	5Ø.ØØ
Ø1/Ø6/83	ZENITH	CASSETTES	25.ØØ
Ø1/Ø6/83	BAKER	SOFTWARE	9Ø.ØØ
Ø1/Ø6/83	BAKER	32K MEMORY	95.ØØ

In this file, the records have been grouped by company name and by the date of sale. We now have two levels of control breaks; our output requires totals for each company and for each date.

Besides the necessity of an extra accumulator and control field, we will also have to decide the order in which the control breaks will be checked. The hierarchy of control breaks must be identified by the programmer; otherwise, the results will be invalid.

The smallest grouping of records should cause minor totals to be displayed. In this case, the totals for a company will show. Date totals could then be called major totals. (If we had more subgroups, they would be known as minor-intermediate and major totals.)

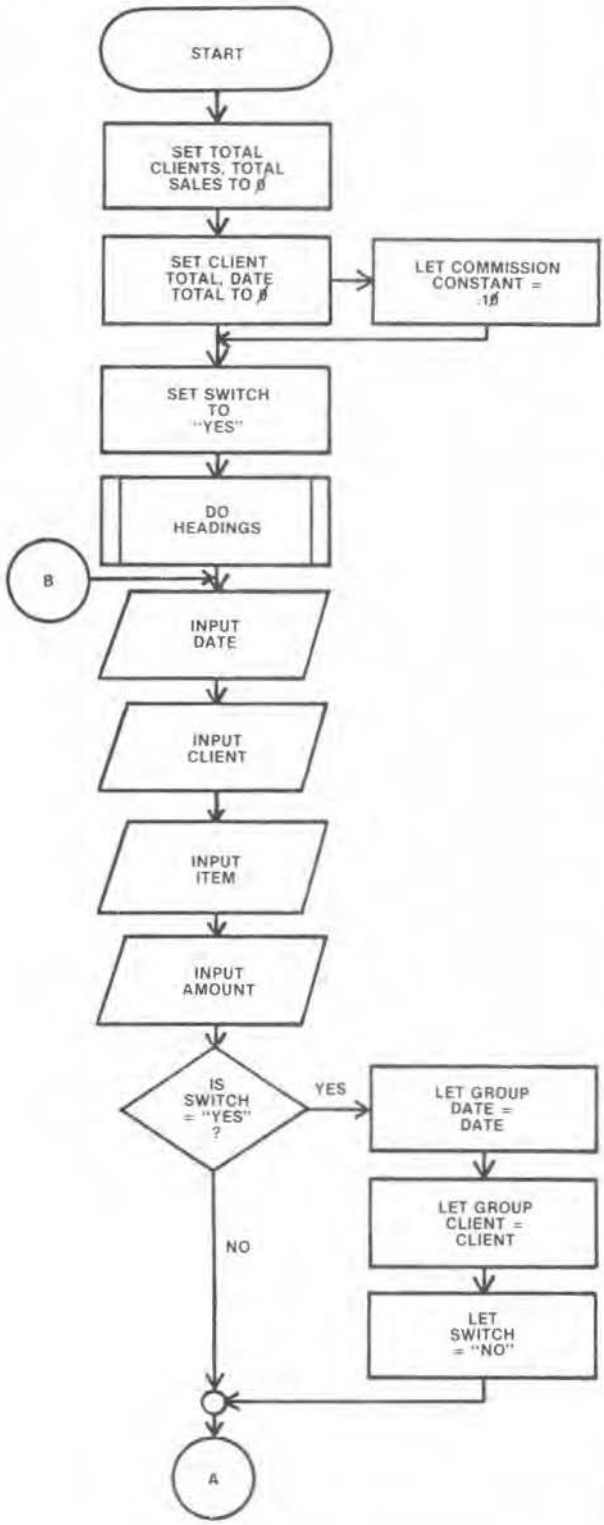
Our output should look like this:



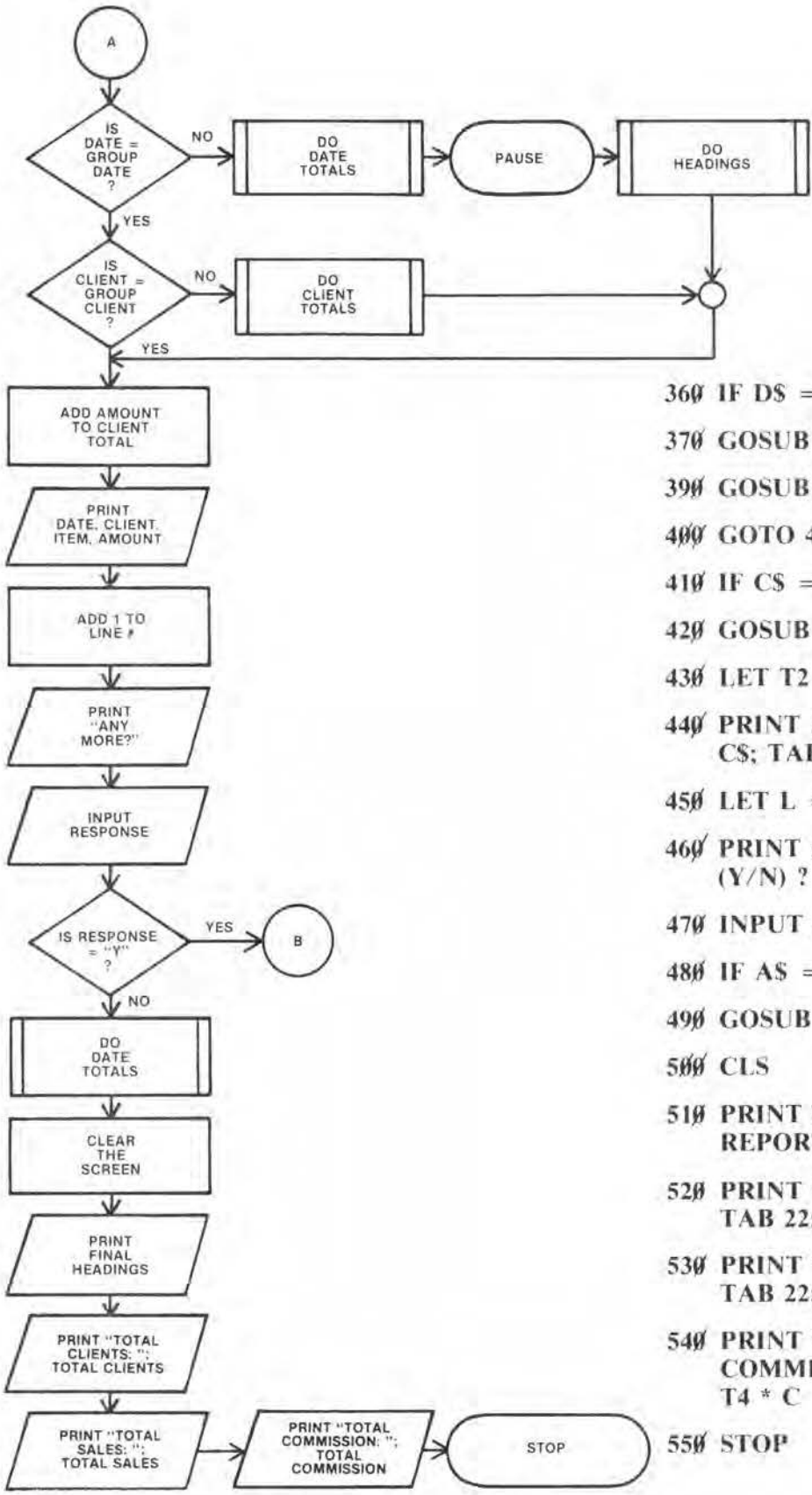
Another screen would then display the next date before our final totals are shown. Note that only the totals for a company are displayed when the company name changes; but when the date changes, the last company totals are printed first and then the date totals.

In the flowcharting design for this application, several subroutines have been incorporated: company totals, date totals and headings.

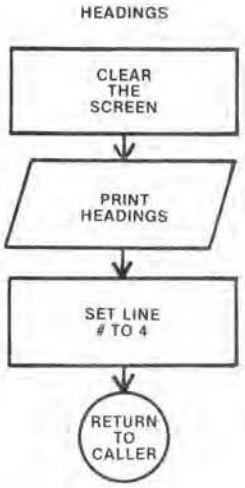
FLOWCHART DESIGN

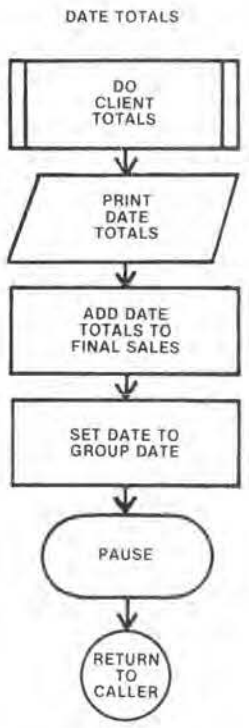


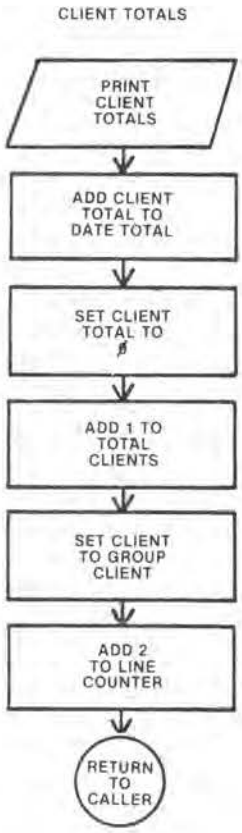

```
1Ø REM IU8A2    MULTIPLE CONTROL BREAKS
2Ø REM D$       DATE OF SALE
3Ø REM C$       CLIENT NAME
4Ø REM I$       ITEM
5Ø REM A        AMOUNT OF SALE
6Ø REM T1       TOTAL CLIENTS
7Ø REM T2       TOTAL SALES — CLIENT
8Ø REM T3       TOTAL SALES — DATE
9Ø REM T4       TOTAL SALES — FINAL
1ØØ REM T5      TOTAL COMMISSION AT .1Ø
11Ø REM L       LINE COUNTER
12Ø REM X$      FIRST RECORD SWITCH
13Ø REM G$      GROUP DATE
14Ø REM P$      GROUP CLIENT
15Ø REM A$      RESPONSE FOR PROMPT
                — ANY MORE?
16Ø REM C       RATE OF COMMISSION
                — .1Ø
17Ø LET T1 = Ø
18Ø LET T2 = Ø
19Ø LET T3 = Ø
2ØØ LET T4 = Ø
21Ø LET C = .1Ø
22Ø LET X$ = "YES"
23Ø GOSUB 57Ø
24Ø PRINT AT 21, Ø; "ENTER
    DATE:"
25Ø INPUT D$
26Ø PRINT AT 21, Ø; "ENTER
    CLIENT:"
27Ø INPUT C$
28Ø PRINT AT 21,Ø; "ENTER
    ITEM : "
29Ø INPUT I$
3ØØ PRINT AT 21, Ø; "ENTER
    AMOUNT:"
31Ø INPUT A
32Ø IF X$<>"YES" THEN GOTO
    36Ø
33Ø LET G$ = D$
34Ø LET P$ = C$
35Ø LET X$ = "NO"
```




```
360 IF D$ = G$ THEN GOTO 410
370 GOSUB 630
390 GOSUB 570
400 GOTO 430
410 IF C$ = P$ THEN GOTO 430
420 GOSUB 700
430 LET T2 = T2 + A
440 PRINT AT L,0; D$; TAB 9;
    C$; TAB 17; I$; TAB 28; A
450 LET L = L + 1
460 PRINT AT 21,0; "MORE
    (Y/N) ?  "
470 INPUT A$
480 IF A$ = "Y" THEN GOTO 240
490 GOSUB 630
500 CLS
510 PRINT TAB 6; "COMMISSION
    REPORT"
520 PRINT "TOTAL CLIENTS: ";
    TAB 22; T1
530 PRINT "TOTAL SALES: ";
    TAB 22; T4
540 PRINT "TOTAL
    COMMISSION: "; TAB 22;
    T4 * C
550 STOP
```








```
560 REM HEADING ROUTINE
570 CLS
580 PRINT TAB 6; "SALES
    ANALYSIS REPORT"
590 PRINT "DATE"; TAB 9;
    "CLIENT"; TAB 19; "ITEM";
    TAB 26; "AMOUNT"
600 LET L = 4
610 RETURN
620 REM DATE TOTALS
630 GOSUB 700
640 PRINT AT L,Ø; "TOTALS
    FOR: "; G$; TAB 25; T3
650 LET T4 = T4 + T3
660 LET T3 = Ø
670 LET G$ = D$
673 PRINT AT 21,Ø; "PRESS ANY
    KEY"
675 PAUSE 32767
680 RETURN
690 REM CLIENT TOTALS
700 PRINT AT L,Ø; "TOTAL
    FOR: "; P$; TAB 25; T2
710 LET T3 = T3 + T2
715 LET T2 = Ø
720 LET T1 = T1 + 1
730 LET P$ = C$
740 LET L = L + 2
750 RETURN
```

Enter the input test data and watch how it runs. The REM statements can be deleted if you wish to conserve memory space. You will see that when you enter a new client's name, totals for the previous client will be displayed. But, when a new date is given, client totals are printed first and then date totals appear. The PAUSE statement allows you to look at the screen before moving on to the next screen. Also, when you respond with "N" to "MORE?," we see first client totals, date totals and, finally, our final totals.

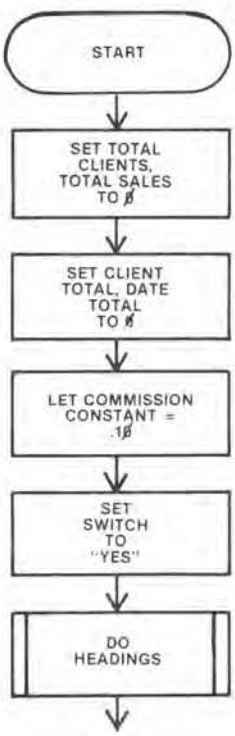
Since this program is very similar to our first program in this Unit, let us now highlight the differences.

ADDITIONAL VARIABLES

Only four variables have been added: an accumulator for client totals (T2), an item description for each record (I$), a comparison area for the group client's name (P$), and a constant for the rate of commission (C).

Several new routines have been added to our basic logic.

INITIALIZATION




```
17Ø LET T1 = Ø
18Ø LET T2 = Ø
19Ø LET T3 = Ø
2ØØ LET T4 = Ø
21Ø LET C = .1Ø
22Ø LET X$ = "YES"
23Ø GOSUB 57Ø
```

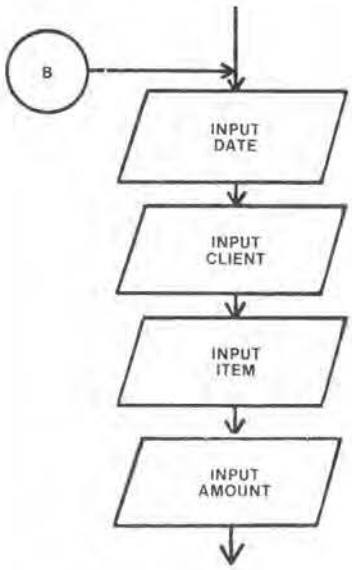
After all accumulators (T1-T4) have been set to Ø, a variable is established as the commission constant of 1Ø% (.1Ø). While not necessary (we didn't use it in the previous program), it is often desirable to utilize this technique of defining "literals" as "constants" in the initialization routine. This allows for easy maintenance of a program when and if this number should change in the future. If, for instance, a commission of 12 percent is to be given, we would have no trouble in merely changing line 21Ø to LET C = .12 and would not have to search through the intricacies of the rest of the program.

Next, our first record bypass switch (X$) is set to "YES," so we will be able to avoid a false control break when we enter data.

In this application, the steps involved in the displaying headings on the top of the screen have been put into a subroutine, since we will have to print a new screen for each new date.

FIGURE 5—Programs for business applications make everyone's job easier. This secretary-bookkeeper used to spend hours figuring out quarterly inventory reports. Now, she has a micro doing most of the work and even allows it to print on her memory typewriter. It is a snap! Can you create a program with control breaks and subroutines for typing paragraphs and letters?

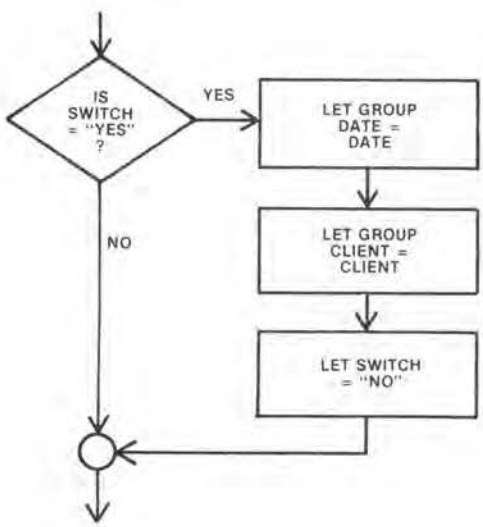
DATA ENTRY ROUTINE




```
24Ø PRINT AT 21,Ø; "ENTER DATE:"
25Ø INPUT D$
26Ø PRINT AT 21,Ø; "ENTER CLIENT:"
27Ø INPUT C$
28Ø PRINT AT 21,Ø; "ENTER ITEM : "
29Ø INPUT I$
3ØØ PRINT AT 21,Ø; "ENTER
    AMOUNT:"
31Ø INPUT A
```

This statement allows for us to enter our detail records, each of which contains four fields: the date of sale, the client's name, the item description, and the amount of the sale.

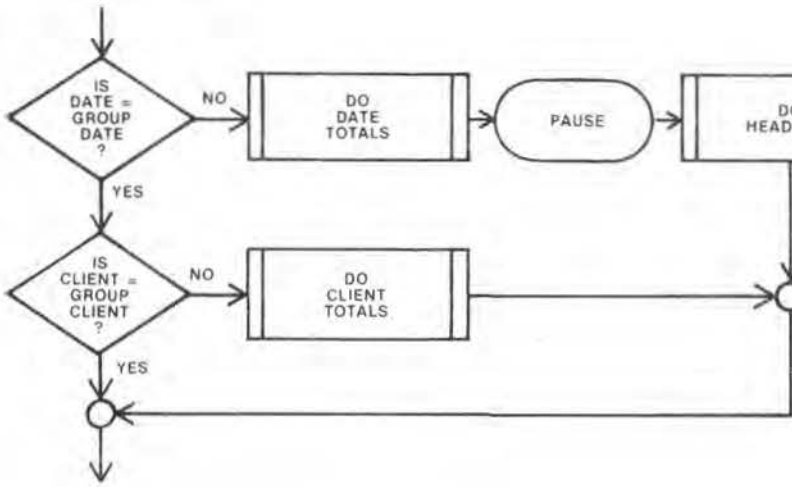
FIRST RECORD BYPASS




```
32Ø IF X$< >"YES" THEN GOTO 36Ø
33Ø LET G$ = D$
34Ø LET P$ = C$
35Ø LET X$ = "NO"
```

In this program, since we have two levels of control breaks, our first record switch has set up a routine which will initialize both control fields before it "turns itself" off.

DETERMINE CONTROL BREAKS




```
36Ø IF D$ = G$ THEN GOTO 41Ø
37Ø GOSUB 63Ø
39Ø GOSUB 57Ø
4ØØ GOTO 43Ø
41Ø IF C$ = P$ THEN GOTO 43Ø
42Ø GOSUB 7ØØ
```

FIGURE 6—Medical data which once required years to collect and analyze now takes a few days or even hours to review and obtain results. Micros and programmers are playing a role in seeking answers to medical problems. Physicians and researchers may request your help in setting up data collection and analysis programs in your community. Can you imagine how loops and subroutines would apply to such program designs?

DO YOU KNOW NOW?

These were the questions posed at the beginning of the lesson.

- **How records can be processed as control groups?**
  You have learned how input data is divided into groups and how programs can be designed to perform intermediate totals. Such applications as total sales per day, commissions of sales persons, or employees' wages earned have been demonstrated in this Study Unit. You have also seen how loops and subroutines are used in such applications.

- **The difference between a GOTO and a GOSUB?**
  A GOTO statement causes a branch with no return. A GOSUB statement causes that particular line number to be held in an area of main storage so that when the RETURN is encountered, the proper branch-back will occur. This area is known as the RETURN STACK.

- **What group printing means?**
  Group printing refers to the displaying of total lines only, and is also known as a summary report. Group printing allows you to receive summarized data without becoming "bogged down" with too many details.

SCHOOL OF COMPUTER TRAINING

EXAM 8

CONTROL BREAKS — TAKING INTERMEDIATE TOTALS

24708-2

Questions 1-10: *Circle the letter beside the one best answer to each question*

1. When utilizing control-break logic, it is assumed that the input data is

   (a) in ascending sequence.
   (b) in descending sequence.
   (c) grouped.
   (d) unable to be processed.

2. Which of the following is *not* essential to group printing?

   (a) A control field.
   (b) A subtotal accumulator.
   (c) Detail lines.
   (d) A first record bypass.

3. A program switch

   (a) is defined and used as the programmer sees fit.
   (b) is a piece of the computer's hardware.
   (c) is always necessary.
   (d) can never be used.

4. The computer program can recognize that control group totals should be printed when

   (a) the last record for a group is entered.
   (b) the first record of a new group is entered.
   (c) a detail record is entered.
   (d) the first record false control-break occurs.

5. A "PAUSE" statement

   (a) terminates a program.
   (b) halts a program until a key is pressed.
   (c) prompts the user to take his time.
   (d) blows a program up.

6. In what way are GOTOs and GOSUBs different?

   (a) Both cause a branch to a line number.
   (b) Both can be part of an IF statement.
   (c) Only a GOSUB causes a branch back when the RETURN is encountered.
   (d) They are alike in all ways.

7. Observe the lines given below and identify which line or lines will enable a prompt to be printed at the bottom of the screen:

```
12Ø CLS
13Ø PRINT TAB 9; "SALES REPORT"
14Ø PRINT TAB 2; "DATE"; TAB 1Ø; "CLIENT"; TAB 2Ø; "AMOUNT"
145 PRINT AT 21,Ø; "ENTER DATE:"
15Ø INPUT D$
155 PRINT AT 21,Ø; "ENTER NAME:"
16Ø INPUT N$
165 PRINT AT 21,Ø; "ENTER SALES:"
17Ø INPUT S
```

   (a) Line 13Ø
   (b) Line 14Ø
   (c) Line 15Ø, 16Ø, 17Ø
   (d) Line 145, 155, 165

8. To ease readability and debugging,

   (a) an additional switch should be entered to any subroutine.
   (b) a remarks line should be added to the beginning of the subroutine.
   (c) a GOSUB statement follows the initial line of the subroutine.
   (d) a RETURN statement is entered at the beginning of the subroutine.

9. The line which is most crucial in designing a subroutine is

   (a) a RETURN statement.
   (b) the STOP statement.
   (c) a PAUSE statement.
   (d) the GOTO statement.

10. After a PAUSE is in effect, you can start the program again by

   (a) turning the computer off, then on again.
   (b) switching the CRT off, then on again.
   (c) touching any key on the keyboard.
   (d) touching only the shift key twice.

WHEN YOU HAVE COMPLETED THE ENTIRE EXAM, TRANSFER YOUR ANSWERS TO THE ANSWER SHEET WHICH FOLLOWS.

ICS

ANSWER PAPER

To avoid delay, please insert all the details requested below

Subject _____ Course _____

Name _____

Address _____

Post Code

Serial Number: 2 4 7 0 8

Test No.: 8

Ed No.: 2

Student's Reference

Letters | Figures

Tutor's Comments

Grade	Tutor

Study the foregoing Question Paper and use it for your rough workings. Record your final answers in the matrix below by writing a cross (X), IN INK OR BALLPOINT, through the letter which you think is the correct answer. Submit ONLY THIS ANSWER SHEET to the School for correction. ALL QUESTIONS MUST BE ANSWERED.

1.	A	B	C	D
2.	A	B	C	D
3.	A	B	C	D
4.	A	B	C	D
5.	A	B	C	D

6.	A	B	C	D
7.	A	B	C	D
8.	A	B	C	D
9.	A	B	C	D
10.	A	B	C	D